(DSPC-8682E)
User Guide

Revision v0.7

Initiated by	Holland Huang Joey Shih	**Job Title**	Supervisor Engineer	**Signature**	
Approved by	Dick Lin	**Job Title**	Software Manager	**Signature**	
2nd Approved		**Job Title**		**Signature**	
Release Status				**Release Date**	

Revision History

Version	Date	Author	Description
0.1	06/19/12	Holland Huang Joey Shih	Initial draft.
0.6	11/16/12	Holland Huang Joey Shih	1. The version number of this document is changed to synchronize with SW package 0.6. 2. The SW package 0.6 support Samsung 4G DDR module. 3. Add subsystem ID and subsystem vendor ID 4. Add I2C boot from address 0x50 to support 64bits address BAR. 5. Add IDT CPS-1616 sRIO switch mode configuration.
0.7	04/03/13	Holland Huang	1. The SW package 0.7 support MCSDK version 2.01.02.05 2. Add individual platform library for DSPC8681 and DSPC8682 in patch of MCSDK 3. Add user mode PCIe driver and DSP local reset function 4. Add DSP init script to support DSP running at 1GHz and 1.25GHz 5. IPC example modification and support PCIe interrupt in DSP SYS/BIOS application

Content

1. Introduction 6
1.1. Hardware Description 6
1.2. DSPC-8682E Block Diagram 6
1.3. DDR3 Interface 7
1.4. PCIe Interface 8
1.5. HyperLink Interface 8
1.6. Serial RapidIO Interface 8
1.7. SGMII Interface 9
1.8. DSP Identification 9
1.9. Hardware Environment Setting 10
2. Package Content 13
2.1. API Interface of DSP Driver 13
2.2. DSP Program Loader Utility 14
2.3. Example: DDR3 Initialization 14
2.4. Example: DSP Initialization for Local Reset 15
2.5. Example: Simple Web Server 15
2.6. Example: PC/DSP Communication 15
2.7. Example: Image Processing 15
2.8. Patch: Platform Library and NDK Library 15
3. DSP Program Loader 17
3.1. Host System Requirement 17
3.2. Build Instruction 17
3.2.1. Build the Driver and Demo Application 17
3.3. Installation and Usage 18

3.4. DSP Loader Utility 19
3.4.1. Query DSP Information 19
3.4.2. Download DSP Program Image 20
3.4.3. DSP Memory Read 21
3.4.4. DSP Memory Write 22
3.4.5. Download DSP Binary File 22
3.4.6. Save DSP Memory as a Binary File 23
3.4.7. DSP Local Reset 24
4. Reference Implementations 25
4.1. Patch of Platform Library and NDK Library 25
4.1.1. How to Use Patch and Pre-built Libraries 25
4.1.2. Build Instruction 25
4.2. DSP DDR3 Initialization 27
4.2.1. Build Instruction 28
4.2.2. Usage 28
4.3. DSP Local Reset 29
4.3.1. Build Instruction 29
4.3.2. Usage 30
4.4. Ethernet and Simple Web Server 30
4.4.1. Build Instruction 30
4.4.2. Usage 31
4.5. Communication between PC and DSP 33
4.5.1. Build Instruction 33
4.5.2. Usage 34
4.5.3. PC Site Utility 35

4.5.4. DSP Demo Program 36
4.5.5. IPC Demo on SYS/BIOS DSP Application 38
4.6. Image Processing Demonstration 39
4.6.1. Build Instruction 40
4.6.2. Usage 40

1. Introduction

This document describes how to set up the software configurations for Octal-DSP PCIe board (DSPC8682E), before using it. The DSPC8682E contains eight Texas Instruments TMS320C6678 DSPs with PCIe, HyperLink, Serial RapidIO, and SGMII interfaces.

1.1. Hardware Description

The placement of the DSPC8682E is shown in Figure 1−1. Each board contains eight TMS320C6678 (codename Shannon) DSPs, one PLX PEX8748 PCIe switch, and one Xilinx XC3S200AN FPGA. The TMS320C6678 multi-core fixed and floating point digital signal processor is based on advanced KeyStone architecture from Texas Instruments. Each TMS320C6678 on DSPC8682E is supported by external DDR3 (the DDR3 module type depends on different HW version) devices for data and program storage. The eight TMS320C6678 devices are connected through PEX8748 PCIe device, which is 48-lane, 12-port PCIe Gen3 switch. The XC3S200AN FPGA device provides the required control signals to the DSPC8682E.

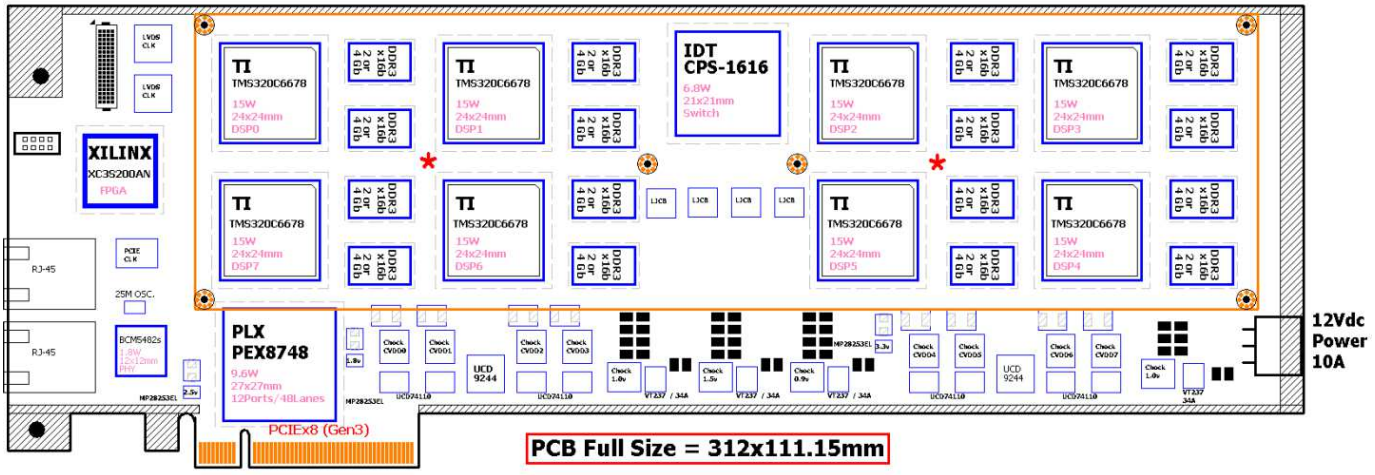

Figure 1-1 DSPC-8682E Placement

1.2. DSPC-8682E Block Diagram

An interface block diagram for the DSPC8682E is shown in Figure 1−2. Each TMS320C6678 DSP contains several interfaces such as DDR, HyperLink, Serial RapidIO, and SGMII.

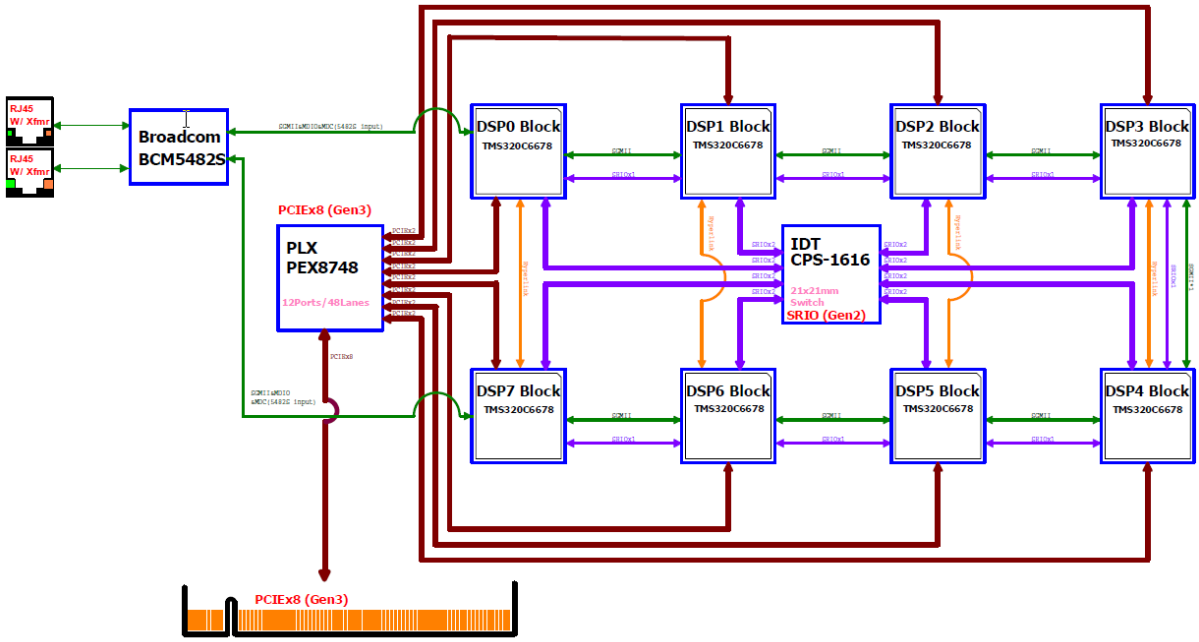

Figure 1-2 DSPC-8682E Interface Block Diagram

1.3. DDR3 Interface

Each TMS320C6678 DSP is connected to four 4Gbit DDR3 memory devices with 64-bit data and 2GB capacity at current implementation. The DDR memory space is ranging from 0x80000000 to 0xFFFFFFFF at DSP device.

Note: The A101 version board is mounted 2Gbit DDR3 memory devices and 1GB capacity. The DDR memory space of those boards will be ranging from 0x80000000 to 0xBFFFFFFF at DSP device.

User can distinguish the HW version on DSPC-8682E backside: 19C2868200 is A101, 19C2868201-01 is A102.

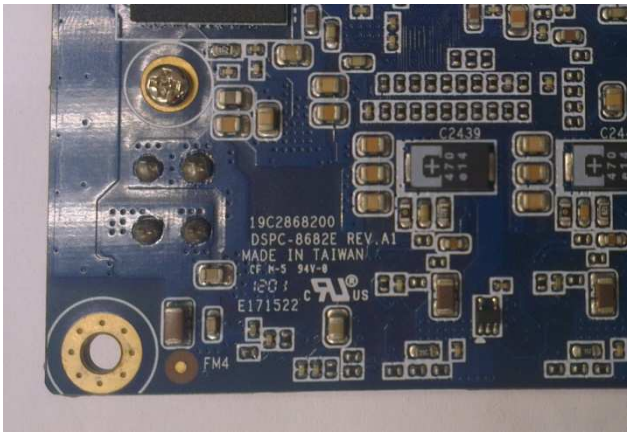

Figure 1-3 Bar Code Label of DSPC-8682E

1.4. PCIe Interface

Each TMS320C6678 DSP is connected to PEX8748 switch by x2-lane of PCIe Gen3 with 5Gb speed per lane. The PEX8748 PCIe switch will connect the DSPC8682E to host PC through x8-lane interface.

1.5. HyperLink Interface

Each pair of TMS320C6678 DSP devices are connected by eight lanes of HyperLink interface with 50Gbaud rate in between. DSP0 and DSP7 is the first DSP pair, DSP1 and DSP6 is the second DSP pair, DSP2 and DSP5 is the third DSP pair and DSP3 and DSP4 is the fourth DSP pair. Each DSP pair can exchange data to link partner via HyperLlink interface as well.

1.6. Serial RapidIO Interface

The DSPC8682E has two ways to communicate to the other DSPs through Serial RapidIO (sRIO) at 5G baud rate. Each TMS320C6678 DSP can use sRIO port0 and port1 to exchange packets with the other DSPs directly in 1x mode or through IDT CPS1616 sRIO switch by port2 to communicate to the other DSPs in 2x mode. The CPS-1616 sRIO switch uses sRIO ID of packets to determine the data flow routing path. The DSP ID and sRIO ID is show below table.

DSP ID	sRIO ID
0	7
1	8
2	4
3	3
4	6
5	5
6	2
7	1

Table 1-1 DSP ID and sRIO ID table

1.7. SGMII Interface

TMS320C6678 DSP contains an on-chip Ethernet switch with two Ethernet interfaces, EMAC0 and EMAC1. TMS320C6678 DSP can connect to another DSP by Ethernet interface without extra Ethernet switch in between. The SGMII interface connection and the topology of the Ethernet link on the DSPC8682E is shown in Figure 1−2. The DSP0 on DSPC8682E contains two SGMII interfaces and EMAC0 is connected to Broadcom BCM5482S Ethernet PHY for external Ethernet access and EMAC1 is connected to EMAC0 of DSP1. EMAC1 of DSP1 is connected to EMAC0 of DSP2. EMAC1 of DSP2 is connecting to EMAC0 of DSP3. EMAC1 of DSP3 is connecting to EMAC0 of DSP4. EMAC1 of DSP4 is connecting to EMAC0 of DSP5. EMAC1 of DSP5 is connecting to EMAC0 of DSP6. The DSP7 EMAC1 is connected to Broadcom BCM5482S Ethernet PHY for external Ethernet access and EMAC0 is connected to EMAC1 of DSP6. Programmers only need to enable Ethernet switch feature of TMS320C6678 DSP and Ethernet packet will forward to the matched DSP by hardware accelerator of on-chip Ethernet switch without intervention of DSP cores inside.

1.8. DSP Identification

The DSPC8682E use GPIO[1:3] pins to identify each DSP and the assignment of DSP ID is shown below:

	GPIO 3	GPIO 2	GPIO 1
DSP 0	0	0	0
DSP 1	0	0	1
DSP 2	0	1	0
DSP 3	0	1	1
DSP 4	1	0	0
DSP 5	1	0	1
DSP 6	1	1	0
DSP 7	1	1	1

Table 1-2 DSP ID and GPIO table

The Linux command “lspci” can list which type of board it is running by checking subsystem ID and subsystem vendor ID as table 1-3.

	SUBSYS_ID	SUBSYS_VEN_ID
Value	0x8682	0x13FE

Table 1-3 Subsystem ID and vendor ID table

```
#lspci -vvnn -d:b005
04:00.0 Multimedia controller [0480]: Texas Instruments Device [104c:b005] (rev
01)
    Subsystem: Advantech Co. Ltd Device [13fe:8682]
    Control: I/O+ Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr-
Stepping- SERR- FastB2B- DisINTx-
    Status: Cap+ 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort- <TAbort-
<MAbort- >SERR- <PERR- INTx-
    Latency: 0, Cache Line Size: 64 bytes
    Interrupt: pin A routed to IRQ 11
    Region 0: Memory at f8800000 (32-bit, non-prefetchable) [size=4K]
    Region 1: Memory at df000000 (32-bit, prefetchable) [size=16M]
    Region 2: Memory at de000000 (32-bit, prefetchable) [size=16M]
    Region 3: Memory at dc000000 (32-bit, prefetchable) [size=32M]
    Region 4: Memory at d8000000 (32-bit, prefetchable) [size=64M]
    Capabilities: <access denied>
```

1.9. Hardware Environment Setting

The DSPC8682E supports two boot modes: Emulation mode and I2C mode. The user can select boot mode by Switch-1 which is shown in Figure 1-4. The Emulation mode is mainly for JTAG debug. The I2C boot mode is usually selected by Switch-1. DSPC-8682E includes eight I2C EEPROMs to support the TMS320C6678 DSPs and each I2C EEPROM contains program for 2-stage boot loader. The 2-stage boot loader will configure PLL and PCIE BAR window when DSP boots up from I2C EEPROM. Table 1-4 and Table 1-5 show the detailed configuration of Switch-1.

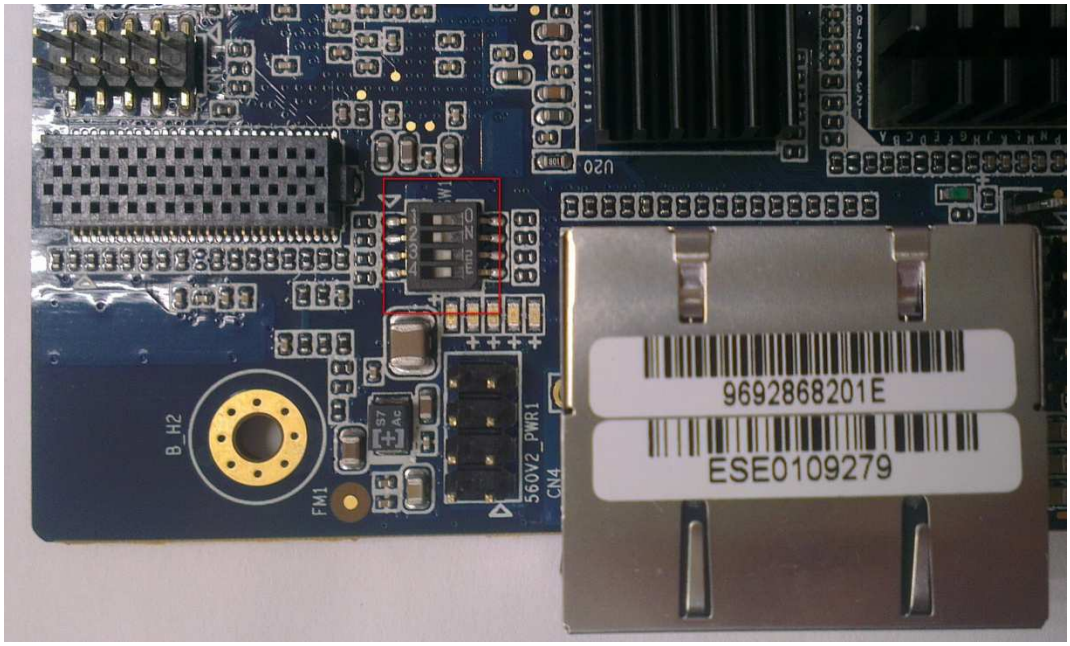

Figure 1-4 I2C Boot Mode (PCIE boot) Setting

Switch 1 pins			
4	3	2	1
Boot mode			Endian

Table 1-4 Switch 1 pin decoding

Bit	Field	Value	Description
4~2	Boot mode	111	Emulation boot mode
		110	I2C boot mode(Boot from address 0x51) 32bits address BAR setting
		100	I2C boot mode(Boot from address 0x50) 64bits address BAR setting
		Others	Reserved
1	Endian	0	Little endian
		1	Big endian

Table 1-5 Switch-1 Configuration Bit Field Description

The Figure 1-5 DIP switch is used to setup sRIO switch mode as below table.

Value	Description
0000	SRIO switch in 2x mode QCFG pins[0:7]=01010101
1111	SRIO switch in 1x mode QCFG pins[0:7]=11111111

Table 1-6 CPS-1616 sRIO switch mode setting

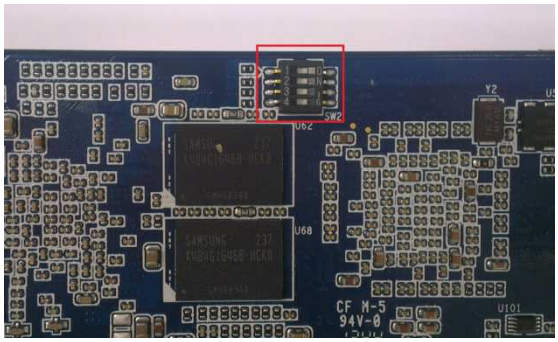

Figure 1-5 4 pins DIP switch to setup IDT-1616 switch mode

CAUSION! It is a known issue when DSPC-8682E boots through secondary boot loader by I2C boot mode the DSP may not complete boot process before BIOS scanning PCIe device tree. Usually DSPC-8682E can be detected after restart BIOS or reboot Linux system.

NOTICE! The Figure 1-6 12V 6pins power connector on DSPC8682E right side must be connecting. If 12V power doesn't connect, the five LEDs should begin flashing to indicate the wrong power status and DSPC-8682E will not boot up.

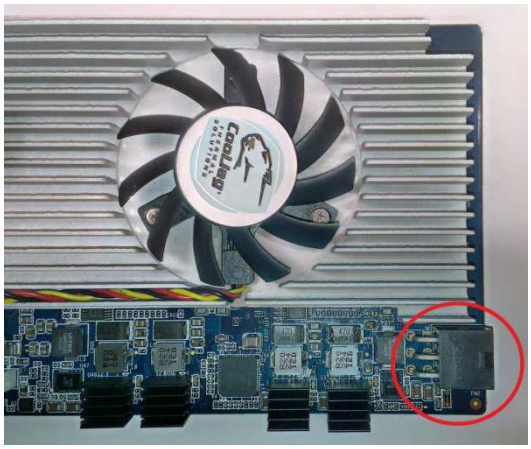

Figure 1-6 The 12V 6pins power connector

2. Package Content

This package is created to help customer quickly boot DSP through PCIE, the package includes:

Path	Purpose
Lightning_PCIE/dsp_loader/driver	DSP Program Loader Driver
Lightning_PCIE/dsp_loader/app	DSP Program Loader Utility
Lightning_PCIE/examples/ddr3	Example: DDR3 Initialization
Lightning_PCIE/examples/dsp_reset	Example: DSP Initialization for Local Reset
Lightning_PCIE/examples/image_processing	Example: Image Processing using Multi-Core
Lightning_PCIE/examples/ipc	Example: PC/DSP Communication
Lightning_PCIE/examples/script	Common demo related scripts
Lightning_PCIE/examples/web	Example: Simple Web Server
Lightning_PCIE/patch	Patch: Platform Library and NDK Library of PDK C6678 1.1.2.5 (inside MCSDK 2.1.2.5)

Table 2-1 Package content list

2.1. API Interface of DSP Driver

It is a Linux based PCIE driver which is used to map between PC memory and DSP memory. A prerequisite for DMA to function is that memory needs to be contiguous in physical memory. Memory allocated using malloc is not contiguous. Ubuntu Linux doesn't have any user mode APIs to allocate contiguous physical memory and hence a Kernel mode driver to allocate contiguous physical memory is necessary. Currently, the implemented I/O controls are listed below:

IOCTL code	Description
TI667X_PCIEEP_IOCTL_GET_BAR_INFO	Get the current BAR information of the specified window
TI667x_PCIEEP_IOCTL_DMA_BUFFER_ALLOC	Allocate buffers of contiguous in physical memory for specified DSP
TI667x_PCIEEP_IOCTL_DMA_BUFFER_FREE	Free all allocated buffers for specified DSP
TI667X_PCIEEP_IOCTL_GET_PCI_INFO	Get PCI Information of DSP

Table 2-2 Kernel Mode Driver I/O Control Code List

A user mode driver is also provided. Developers can implement their own application based on this user mode PCIe driver. The APIs are listed below:

Export API	Description
pcie_drv_open	Open the devices which are registered by kernel driver, and set up the access of PCIe BAR regions
pcie_drv_close	Close the devices and free all allocated resources
pcie_drv_set_ep_config	Set PCIe endpoint related configurations, such as interrupt and privilege register.
pcie_drv_dsp_set_entry_point	Write the entry point to boot magic address. The boot magic address is the lasted DWORD of L2 memory, for C6678, the address is 0x0087FFFC
pcie_drv_dsp_write	Write to DSP memory using memcpy over PCIe
pcie_drv_dsp_read	Read from DSP memory using memcpy over PCIe
pcie_drv_dma_mem_alloc	Allocate contiguous host memory for specified DSP
pcie_drv_dma_mem_free	Free the allocated physical memory and unmap all host memory for specified DSP
pcie_drv_dma_write	Write data to DSP memory from provided contiguous host memory
pcie_drv_dma_read	Read data from DSP memory to provided contiguous host memory
pcie_drv_dsp_int_select	Wait interrupt signal from DSP
pcie_drv_get_dsp_dev_info	Get PCIe information of all DSP devices

Table 2-3 User Mode Driver API List

2.2. DSP Program Loader Utility

DSP program loader utility contains a hex parser and is used to load hex files into DSPs and notify DSPs to run program.

2.3. Example: DDR3 Initialization

The DDR3 initialization example contains CCS project settings to build a boot image. This program will initialize DDR by reading parameter that stored in EEPROM(address 0x51, offset 65500 bytes) after software 0.6 release, otherwise the Samsung 2Gb and Micron 4Gb

DDR module initialization parameter are hard code setting, DSP will wait loader utility to load the next program after DDR initialization finished. A file format conversion tool provided by TI is also included and can be used to convert .out file format into .hex file format.

2.4. Example: DSP Initialization for Local Reset

The DSP reset example contains CCS project settings to build a boot image. This program is a part of DSP local reset procedure. By running this program, core0 will poll PCIe legacy INTA that is generated from host. Other cores will enter idle state after local reset related registers are set by DSP Program Loader.

2.5. Example: Simple Web Server

A web demo example contains CCS project settings to build an image. It can set up a web server so user can use network browser to access the web page stored in the DSP. This program is modified from TI MCSDK example which is located in the mcsdk_2_01_02_05\examples\ndk\client. The each DSP will be configured with a static IP instead of DHCP.

2.6. Example: PC/DSP Communication

This example contains two parts, a DSP image and a PC utility. The dsp folder included contains CCS project settings of building an image. This example provides sample codes on how to communicate between PC and DSP.

2.7. Example: Image Processing

The image processing demo example contains two CCS project settings to build the demo images. This application will run TI image processing kernels (imagelib) on multiple cores to do image processing (eg: edge detection, etc) on an input image. This program is modified from TI MCSDK example which is located in the mcsdk_2_01_02_05\demos\image_processing\ipc. The each DSP will be configured with a static IP instead of DHCP.

2.8. Patch: Platform Library and NDK Library

There are some differences between the DSPC8682E and C6678 EVM, hence, developer should patch these files in the TI PDK before using it. The modification is listed as below:

1. The DSPC8682E uses DSP0 EMAC0 and DSP7 EMAC1 to connect to BCM5482S Ethernet PHY. This patch adds the initialization of SGMII port0 and change settings of SGMII port 0 and port 1 for BCM5482S Ethernet PHY.

2. DSPC-8682E uses different DDR memory devices, the parameters for initialization of DDR controller is not the same as C6678 EVM. This patch supports the DDR memory device which is mounted on DSPC-8682E.
3. The reference clocks of DDR and SGMII is not the same as C6678 EVM and this patch modifies the relevant MPY settings.

3. DSP Program Loader

After the whole system booting up, all DSP chips stay in idle mode. The PC is responsible to download DSP codes to every chip and awaken DSPs to execute the loaded codes. The loader consists of a driver and a utility running in PC Linux environment. This package contains source code of the program loader. The developer must rebuild and install them to the Linux before starting using the Lightning board.

3.1. Host System Requirement

A reference of the OS used to develop and execute this software release is:

1. **Linux distribution**: Ubuntu 10.10. Other distributions including Debian, Redhat, CentOS, and Fedora should work with this software package.
2. **Kernel**: Linux kernel version 2.6.35.22. In fact, the driver should work with any kernel with version >=2.6.20.
3. **Pre-required Library**: libreadline5-dev.deb or libreadline5-dev.rpm for Redhat families.
4. **DSP development tool**: TI Code Composer Studio v5.1 or higher, TI MCSDK for TMS320C66x Processors V2.01.02.05, please refer to web site: http://software-dl.ti.com/sdoemb/sdoemb_public_sw/bios_mcsdk/02_01_02_05/index_FDS.html

3.2. Build Instruction

3.2.1. Build the Driver and Demo Application

The driver is closely tied to Linux kernel running on PC, therefore, it must be rebuilt to work with the supporting kernel. The commands for building PCIE driver are listed below:

```
# /Lightning_PCIE/make clean
# /Lightning_PCIE/make
```

This compiles the PCIe kernel driver, user mode driver, dsp_loader utility and pc site application of ipc example. The Module libraries can be found in the dsp_loader/driver/module directory. The user mode driver library will be generated in the dsp_loader/driver/lib directory. The dsp_loader executable can be found in dsp_loader/app/bin directory. The pc site application dsp_demo executable will be produced in the examples/ipc/pc/bin.

3.3. Installation and Usage

Linux host PCIE driver is used to create mapping between PC memory and DSP memory. Users can run the shell script “load.sh” to load and install the driver. The script “unload.sh” is used to unload the driver.

```
# /Lightning_PCIE/dsp_loader/driver/module/sh load.sh
# /Lightning_PCIE/dsp_loader/driver/module/sh unload.sh
```

The device information is shown by dmesg command.

```
# /Lightning_PCIE/dsp_loader/driver/module/dmesg

dspc868x_pcie_ep: Found TI667x PCIe EP @0xffff880073ab6000
dspc868x_pcie_ep: Found TI667x PCIe EP @0xffff880073ab7000
dspc868x_pcie_ep: Found TI667x PCIe EP @0xffff880073510000
dspc868x_pcie_ep: Found TI667x PCIe EP @0xffff880073511000
dspc868x_pcie_ep: Found TI667x PCIe EP @0xffff880073512000
dspc868x_pcie_ep: Found TI667x PCIe EP @0xffff880073513000
dspc868x_pcie_ep: Found TI667x PCIe EP @0xffff880073514000
dspc868x_pcie_ep: Found TI667x PCIe EP @0xffff880073515000
dspc868x_pcie_ep: detect 8 DSP in this system
pci 0000:03:00.0: Major 249 Minor 0 assigned
pci 0000:03:00.0: Added device to the sys file system
pci 0000:03:00.0:       BAR Configuration:
pci 0000:03:00.0:       Start          |      Length         |      Flags
pci 0000:03:00.0:       0xfb7ff000      |       00004096       |       0x00040200
pci 0000:03:00.0:       0xa7000000      |       16777216       |       0x00042208
pci 0000:03:00.0:       0xa6000000      |       16777216       |       0x00042208
pci 0000:03:00.0:       0xa4000000      |       33554432       |       0x00042208
pci 0000:03:00.0: TI667X registers mapped to 0xffffc9000035e000
pci 0000:04:00.0: Major 249 Minor 1 assigned
pci 0000:04:00.0: Added device to the sys file system
pci 0000:04:00.0:       BAR Configuration:
pci 0000:04:00.0:       Start          |      Length         |      Flags
pci 0000:04:00.0:       0xfb8ff000      |       00004096       |       0x00040200
pci 0000:04:00.0:       0xaf000000      |       16777216       |       0x00042208
pci 0000:04:00.0:       0xae000000      |       16777216       |       0x00042208
pci 0000:04:00.0:       0xac000000      |       33554432       |       0x00042208
pci 0000:04:00.0: TI667X registers mapped to 0xffffc9000037e000
pci 0000:05:00.0: Major 249 Minor 2 assigned
pci 0000:05:00.0: Added device to the sys file system
pci 0000:05:00.0:       BAR Configuration:
pci 0000:05:00.0:       Start          |      Length         |      Flags
pci 0000:05:00.0:       0xfb9ff000      |       00004096       |       0x00040200
pci 0000:05:00.0:       0xb7000000      |       16777216       |       0x00042208
pci 0000:05:00.0:       0xb6000000      |       16777216       |       0x00042208
pci 0000:05:00.0:       0xb4000000      |       33554432       |       0x00042208
pci 0000:05:00.0: TI667X registers mapped to 0xffffc9001097e000
pci 0000:06:00.0: Major 249 Minor 3 assigned
pci 0000:06:00.0: Added device to the sys file system
pci 0000:06:00.0:       BAR Configuration:
pci 0000:06:00.0:       Start          |      Length         |      Flags
```

```
pci 0000:06:00.0:       0xfbaff000      |       00004096        |       0x00040200
pci 0000:06:00.0:       0xbf000000      |       16777216        |       0x00042208
pci 0000:06:00.0:       0xbe000000      |       16777216        |       0x00042208
pci 0000:06:00.0:       0xbc000000      |       33554432        |       0x00042208
pci 0000:06:00.0: TI667X registers mapped to 0xffffc90010a16000
pci 0000:07:00.0: Major 249 Minor 4 assigned
pci 0000:07:00.0: Added device to the sys file system
pci 0000:07:00.0:       BAR Configuration:
pci 0000:07:00.0:       Start           |       Length          |       Flags
pci 0000:07:00.0:       0xfbbff000      |       00004096        |       0x00040200
pci 0000:07:00.0:       0xc7000000      |       16777216        |       0x00042208
pci 0000:07:00.0:       0xc6000000      |       16777216        |       0x00042208
pci 0000:07:00.0:       0xc4000000      |       33554432        |       0x00042208
pci 0000:07:00.0: TI667X registers mapped to 0xffffc90010a3e000
pci 0000:08:00.0: Major 249 Minor 5 assigned
pci 0000:08:00.0: Added device to the sys file system
pci 0000:08:00.0:       BAR Configuration:
pci 0000:08:00.0:       Start           |       Length          |       Flags
pci 0000:08:00.0:       0xfbcff000      |       00004096        |       0x00040200
pci 0000:08:00.0:       0xcf000000      |       16777216        |       0x00042208
pci 0000:08:00.0:       0xce000000      |       16777216        |       0x00042208
pci 0000:08:00.0:       0xcc000000      |       33554432        |       0x00042208
pci 0000:08:00.0: TI667X registers mapped to 0xffffc90010a7c000
pci 0000:09:00.0: Major 249 Minor 6 assigned
pci 0000:09:00.0: Added device to the sys file system
pci 0000:09:00.0:       BAR Configuration:
pci 0000:09:00.0:       Start           |       Length          |       Flags
pci 0000:09:00.0:       0xfbdff000      |       00004096        |       0x00040200
pci 0000:09:00.0:       0xd7000000      |       16777216        |       0x00042208
pci 0000:09:00.0:       0xd6000000      |       16777216        |       0x00042208
pci 0000:09:00.0:       0xd4000000      |       33554432        |       0x00042208
pci 0000:09:00.0: TI667X registers mapped to 0xffffc90010a7e000
pci 0000:0a:00.0: Major 249 Minor 7 assigned
pci 0000:0a:00.0: Added device to the sys file system
pci 0000:0a:00.0:       BAR Configuration:
pci 0000:0a:00.0:       Start           |       Length          |       Flags
pci 0000:0a:00.0:       0xfbeff000      |       00004096        |       0x00040200
pci 0000:0a:00.0:       0xdf000000      |       16777216        |       0x00042208
pci 0000:0a:00.0:       0xde000000      |       16777216        |       0x00042208
pci 0000:0a:00.0:       0xdc000000      |       33554432        |       0x00042208
pci 0000:0a:00.0: TI667X registers mapped to 0xffffc90010e92000
```

3.4. DSP Loader Utility

DSP loader offers the functions to load the program into DSP memory and notify the DSP to run program.

3.4.1. Query DSP Information

The command syntax is:

dsp_loader query list or dsp_loader query -l

dsp_loader query [chip#]

The command is to display the PCI information of DSP which are installed in the system. The more detailed information will be displayed when user specify the **[chip#]** parameter. The [chip#] are the number of DSPs attached to the PC. Since there are eight DSP devices on the DSPC8682E, this parameter can be set into 0 ~ 7 for those PC systems installed with one DSPC8682E card. For those PC systems installed with two DSPC8682E cards, there will be sixteen chips available to the PC systems and the parameter can be set into 0 ~ 15.

The following two examples demonstrate the result of query command when PC system install DSPC8682E card and query the detailed information of DSP#7.

```
# /Lightning_PCIE/dsp_loader/app/bin/dsp_loader query list

Card 0:
    [Chip 0] Device 8682
    [Chip 1] Device 8682
    [Chip 2] Device 8682
    [Chip 3] Device 8682
    [Chip 4] Device 8682
    [Chip 5] Device 8682
    [Chip 6] Device 8682
    [Chip 7] Device 8682

# /Lightning_PCIE/dsp_loader/app/dsp_loader query 7
=============================================
Chip: 7

    PCI Bridge 2
    PCI Bus Num: 10
    Vendor ID: 0x104c    Device ID: 0xb005
    Subsystem VendorID: 0x13fe     Subsystem DevID: 0x8682
    Class: 0x00048000
    Header Type: 0  Irq Pin: 1
    BAR Configuration:
    Start        |   Length       |   Flags
    0xfbeff000   |   00004096|   0x00020200
    0xdf000000   |   16777216|   0x00021208
    0xde000000   |   16777216|   0x00021208
    0xdc000000   |   33554432|   0x00021208
=============================================
```

3.4.2. Download DSP Program Image

The command syntax is:

dsp_loader load [chip#] [core#] [image entry point] [image file name (hex)]

The command is to download a DSP program (DSP image) into to RAM of a specified DSP. The detailed description of each parameter is shown below:

1. **[chip#]:** the [chip#] are the number of DSPs attached to the PC.
2. **[core#]:** [core#] is used to notify individual core (range from 0 to 7) within DSP to run.
3. **[image entry point]:** [image entry point] is the start address of the loaded image. User can find the “entry point symbol” of "_c_int00" in the map file. For example, init.map information is displayed in List 3-1. The reader can find the entry point of the program in the top of map file.

```
******************************************************************************
                  TMS320C6x Linker PC v7.3.1
******************************************************************************
>> Linked Fri Jun 15 11:12:43 2012


OUTPUT FILE NAME:      <../bin/init.out>
ENTRY POINT SYMBOL: "_c_int00"    address: 0083b900
```

List 3-1 entry point in the init.map

4. **[image file name]:** [image file name] is the full path of hex file name which is loaded into DSP.

The following example demonstrates how to load /Lightning_PCIE/bin/DSPC8682E/init.hex (DSP image for DDR initialization) into DSP#1 and use CPU#0 to run DSP image.

```
# /Lightning_PCIE/dsp_loader/app/bin/dsp_loader load 1 0 0x0083b900
/Lightning_PCIE/bin/DSPC8682E/init.hex

Load HEX image: /Lightning_PCIE/bin/DSPC8682E/init.hex to 1:0, start address
0x0083b900
Load HEX OK
```

Note: Image entry point depends on DSP image. The image entry point of init.hex (DSP image) uses address 0x0083B900 as local address for each CPU. Individual local CPU address can also be transferred to DSP global address with offset. For example, CPU#0 local address 0x00800000 is equal to DSP global address 0x1080000. CPU#1 local address 0x00800000 is equal to DSP global address 0x1180000.

3.4.3. DSP Memory Read

The command syntax is:

dsp_loader rmem [chip#] [address]

The command is to read a 32bits-DWORD from DSP. The detailed description of each parameter is shown below:

1. **[chip#]:** the [chip#] are the number of DSPs attached to the PC.
2. **[address]**: read data address

The following example is to read DSP#2 data at address 0x10800000.

```
# /Lightning_PCIE/dsp_loader/app/bin/dsp_loader rmem 2 0x10800000
0x01bc54f6
```

3.4.4. DSP Memory Write

The command syntax is:

dsp_loader load [chip#] [address][value]

The command is to write a 32bits-DWORD into DSP memory. The detailed description of each parameter is shown below:

1. **[chip#]:** the [chip#] are the number of DSPs attached to the PC.
2. **[address]**: written data address
3. **[value]**: written data

The following example writes data 0x55AA55AA into DSP#2 at address 0x10800000.

```
# /Lightning_PCIE/dsp_loader/app/bin/dsp_loader wmem 2 0x10800000 0x55aa55aa

# /Lightning_PCIE/dsp_loader/app/bin/dsp_loader rmem 2 0x10800000
0x55aa55aa
```

3.4.5. Download DSP Binary File

The command syntax is:

dsp_loader loadbinary [chip#][address][size][transfer type][bin file name]

The command is to write a bin file into DSP memory. The detailed description of each parameter is shown below:

1. **[chip#]:** the [chip#] are the number of DSPs attached to the PC.
2. **[address]**: written data address
3. **[size]**: written data size, 0 for all data of file.
4. **[transfer type]**: 0 for CPU memcpy, 1 for DMA.

5. **[bin file name]**: [bin file name] is the full path of binary file name which is loaded into DSP.

The following example writes a jpg file into DSP#1 at DDR beginning address 0x80000000 by using DMA.

```
# /Lightning_PCIE/dsp_loader/app/bin/dsp_loader loadbinary 1 0x80000000 0 1
/home/advantech/test_image.jpg
Load Binary file: /home/advantech/test_image.jpg to DSP1, start address
0x80000000, Size 0x00000000
Written to dsp 7496169 bytes
Time measured: 16225 us
Load Binary OK
```

3.4.6. Save DSP Memory as a Binary File

The command syntax is:

dsp_loader savebinary [chip#][address][size][transfer type][bin file name]

The command is to read a DSP memory section and save the data as a binary file. The detailed description of each parameter is shown below:

1. **[chip#]:** the [chip#] are the number of DSPs attached to the PC.
2. **[address]**: read data address
3. **[size]**: read data size
4. **[transfer type]**: 0 for CPU memcpy, 1 for DMA.
5. **[bin file name]**: [bin file name] is the full path of binary file name which is saved.

The following example read 7496169 bytes from DSP#1 at DDR beginning address 0x80000000 by using DMA, and saves as test_image_output.jpg.

```
# /Lightning_PCIE/dsp_loader/app/bin/dsp_loader loadbinary 1 0x80000000 0 1
/home/advantech/test_image.jpg
Load Binary file: /home/advantech/test_image.jpg to DSP1, start address
0x80000000, Size 0x00000000
Written to dsp 7496169 bytes
Time measured: 16225 us
Load Binary OK

# /Lightning_PCIE/dsp_loader/app/bin/dsp_loader savebinary 1 0x80000000 7496169
1 /home/advantech/test_image_output.jpg
Save Binary file: /home/advantech/test_image_output.jpg from DSP 1, start
address 0x80000000 Size 0x007261e9
Saved from dsp 7496169 bytes
Time measured: 21871 us
Save Binary OK
```

3.4.7. DSP Local Reset

The command syntax is:

dsp_loader reset [chip#]

The command is to do a local reset of DSP. The detailed description of each parameter is shown below:

1. **[chip#]:** the [chip#] are the number of DSPs attached to the PC.

The following example reset DSP#0.

```
# /Lightning_PCIE/dsp_loader/app/bin/dsp_loader reset 0

 Iterations waited for entry point to clear 1
Dsp 0:  DSP Reset success !
```

4. Reference Implementations

4.1. Patch of Platform Library and NDK Library

The example programs have to link with DSPC8682 platform library and NDK library. A developer has to install MCSDK first and applies the provided patch. The default path of MCSDK in Windows is "C:\Program Files\Texas Instruments\" or "C:\ti".

4.1.1. How to Use Patch and Pre-built Libraries

1. Copy Lightning_PCIE\patch\pdk_C6678_1_1_2_5.
2. Paste to C:\Program Files\Texas Instruments\pdk_C6678_1_1_2_5.
3. The pre-built libraries are included in the provided patch, a developer can use these libraries directly.

4.1.2. Build Instruction

Steps to build platform lib are listed below:

1. Import the CCS project from "pdk_C6678_1_1_2_5\packages\ti\platform\dspc8682\platform_lib" directory (in CCSv5, Project->Import Existing CCS/CCE Eclipse Projects)
2. Refer to Figure 4-1 ~ Figure 4-3 and select "Lite" as active configuration (in CCSv5, Project->Properties)

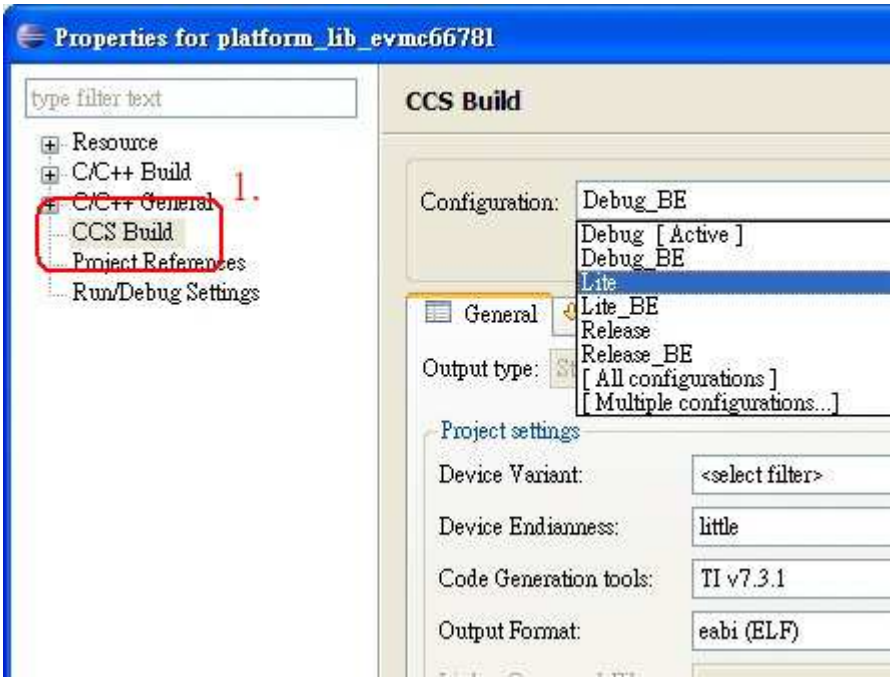

Figure 4-1 Select Lite as Active Configuration (Step 1)

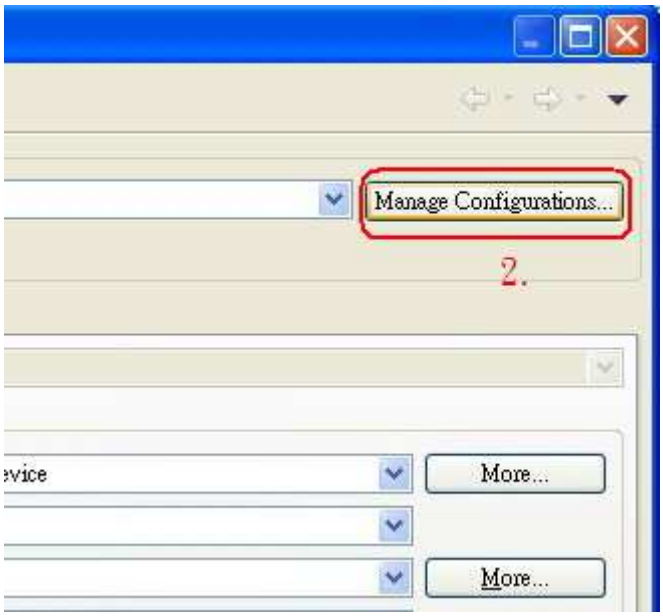

Figure 4-2 Select Lite as Active Configuration (Step 2)

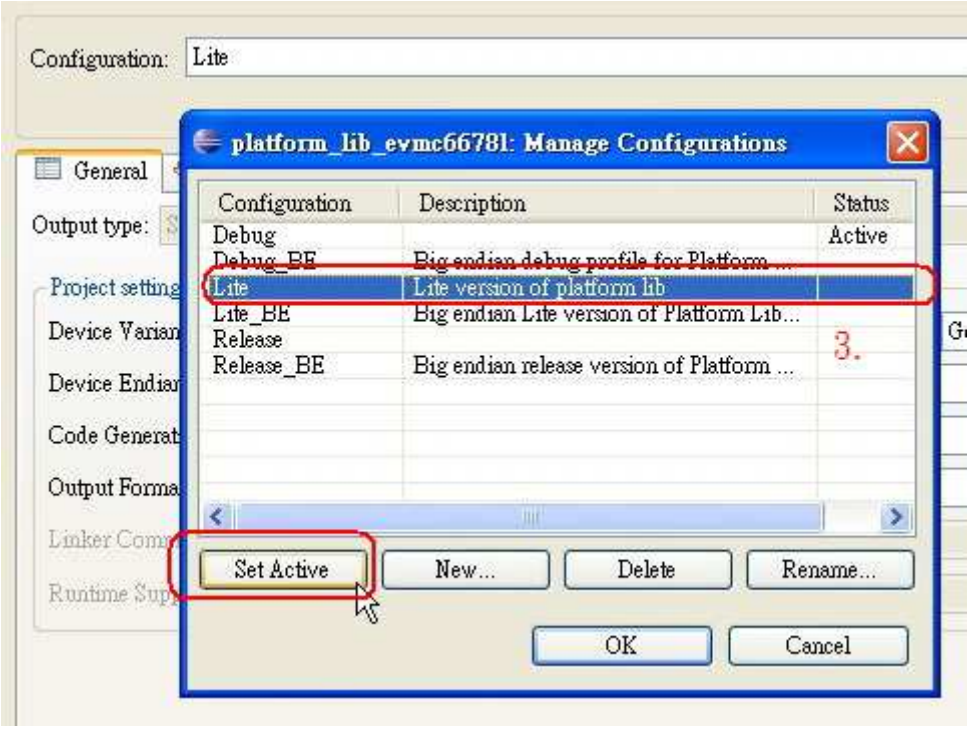

Figure 4-3 Select Lite as Active Configuration (Step 2)

3. Clean the platform_lib_dspc8682 project and re-build the project. After build is completed the ti.platform.dspc8682.lite.lib will be generated under the directory: "pdk_C6678_1_1_2_5\packages\ti\platform\dspc8682\platform_lib\lib\debug"

4. Repeat step 2 and step 3, select "Debug" as active configuration and re-build the project. ti.platform.dspc8682.ae66 will be generated under the same directory

Steps to build ndk lib are listed below:

1. Import the CCS project from "pdk_C6678_1_1_2_5\packages\ti\transport\ndk\nimu" directory (in CCSv5, Project->Import Existing CCS/CCE Eclipse Projects)

2. Clean the nimu_eth_evmc6678l project and re-build the project. After build is completed, ti.transport.ndk.nimu.ae66 will be generated under the directory: "pdk_C6678_1_1_2_5\packages\ti\transport\ndk\nimu\lib\debug"

4.2. DSP DDR3 Initialization

The Boot ROM codes only initialize L2 internal memory when booting from PCIE boot mode. The on-board DDR3 control registers need to be explicitly initialized by this supplied example program. User has to initialize DDR3 control registers before loading the application into DSP. After initialization of DDR3 control registers, this program will clear boot address and

wait dsp_loader to write new entry point in boot address. When boot address is updated, this program will jump to new entry point and start to run the next program.

4.2.1. Build Instruction

Steps to build DDR3 initialization program are listed below:

1. Import the demo_evmc6678l_init CCS project from "Lightning_PCIE\examples\ddr3\evmc6678l" directory (in CCSv5, Project->Import Existing CCS/ CCE Eclipse Projects)
2. Select DSPC8682E as active configuration
3. Clean the demo_evmc6678l_init project and re-build the project. After build is completed, init.out and init.map will be generated under "Lightning_PCIE\examples\ddr3\evmc6678l\bin\DSPC8682E" directory

4.2.2. Usage

User can use the shell script examples\script\DSPC8682E\init_1000.sh or init_1250.sh to initialize DSP DDR, the procedure is composed of three jobs,

1. Convert .out to .hex (by executable Hex6x)
2. Externally set PLL Multiplier configuration (by dsp_loader)
3. Load .hex to DSP (by dsp_loader)

There are two scripts, init_1000.sh and init_1250.sh for users to initialize DSP. They load the init.hex in Lightning_PCIE\bin. DSP initialization can be done by invoking the init scripts. The difference between init_1000.sh and init_1250.sh is that the former runs DSP at 1GHz, the latter overclocks DSP to 1.25GHz. There is the prebuilt binary bundled in Lightning_PCIE\bin. Users can initialize DSP DDR module without building the image from source. However, when running the script, it will show the version of your DSP (PG1 or PG2), and for PG2 chip, it will also show the maximum running frequency, e.g., 1GHz, 1.2GHz, or 1.25GHz.

Notice: At present, we only guarantee the stability for PG2 version of C6678 to run at 1GHz.

The following example initializes 4 DSP.

```
# /Lightning_PCIE/examples/script/DSPC8682E/init_1000.sh 8

Translating to Intel format...
   "../../ddr3/evmc6678l/bin/DSPC8682E/init.out"   ==> .text:_c_int00
   "../../ddr3/evmc6678l/bin/DSPC8682E/init.out"   ==> .text
   "../../ddr3/evmc6678l/bin/DSPC8682E/init.out"   ==> .const
   "../../ddr3/evmc6678l/bin/DSPC8682E/init.out"   ==> .cinit
DSP type: silicon Version = PG2.0, DSP maximum frquency limit = 1GHz
Load HEX image: ../../../bin/DSPC8682E/init.hex to 0:0, start address 0x00830000
Load HEX OK
```

```
DSP type: silicon Version = PG2.0, DSP maximum frquency limit = 1GHz
Load HEX image: ../../../bin/DSPC8682E/init.hex to 1:0, start address 0x00830000
Load HEX OK
DSP type: silicon Version = PG2.0, DSP maximum frquency limit = 1GHz
Load HEX image: ../../../bin/DSPC8682E/init.hex to 2:0, start address 0x00830000
Load HEX OK
DSP type: silicon Version = PG2.0, DSP maximum frquency limit = 1GHz
Load HEX image: ../../../bin/DSPC8682E/init.hex to 3:0, start address 0x00830000
Load HEX OK
DSP type: silicon Version = PG2.0, DSP maximum frquency limit = 1GHz
Load HEX image: ../../../bin/DSPC8682E/init.hex to 4:0, start address 0x00830000
Load HEX OK
DSP type: silicon Version = PG2.0, DSP maximum frquency limit = 1GHz
Load HEX image: ../../../bin/DSPC8682E/init.hex to 5:0, start address 0x00830000
Load HEX OK
DSP type: silicon Version = PG2.0, DSP maximum frquency limit = 1GHz
Load HEX image: ../../../bin/DSPC8682E/init.hex to 6:0, start address 0x00830000
Load HEX OK
DSP type: silicon Version = PG2.0, DSP maximum frquency limit = 1GHz
Load HEX image: ../../../bin/DSPC8682E/init.hex to 7:0, start address 0x00830000
Load HEX OK
```

4.3. DSP Local Reset

After DSP code is downloaded once, the DSP runs downloaded code. In order to re-download the different DSP code, the DSP local reset function is needed. When user perform the reset function by DSP loader utility, the utility will configure related registers of each module of DSP, and download the DSP reset program to each core. The file format conversion tool is also included and can be used to convert .out file format into .h file, which will be used as the source file when make DSP loader utility.

4.3.1. Build Instruction

Steps to build DSP reset program are listed below:

1. Import the pcieboot_localreset CCS project from "Lightning_PCIE\examples\dsp_reset\build" directory (in CCSv5, Project->Import Existing CCS/ CCE Eclipse Projects)
2. Clean the pcieboot_localreset project and re-build the project. After build is completed, pcieboot_localreset.out and pcieboot_localreset.map will be generated under "Lightning_PCIE\examples\ dsp_reset\build\bin" directory
3. Enter in "Lightning_PCIE\examples\script\utils\elf2HUtils" and launch pcieboot_localreset_elf2HBin.sh, After the steps of script are completed, the pcieLocalReset.h will be generated under "Lightning_PCIE\dsp_loader\app\inc"
4. Enter in "Lightning_PCIE\", make clean and make (follow chap 3.2) to re-build DSP loader.

4.3.2. Usage

Refer to 3.4.7 to get detailed procedure of the DSP local reset.

4.4. Ethernet and Simple Web Server

The Ethernet program is modified from the example codes in TI MCSDK. This example implements a simple web server running on DSP. The Ethernet port on the bracket of the DSPC8682E must be connected to an external Ethernet switch (support gigabit rate) before running this example. Each DSP has a fixed IP number that is determined by its order. The pre-given IP addresses are shown below. The user can use a browser to view the simple web page provided by this simple web server.

	IP
DSP 0	192.168.1.101
DSP 1	192.168.1.102
DSP 2	192.168.1.103
DSP 3	192.168.1.104
DSP 4	192.168.1.105
DSP 5	192.168.1.106
DSP 6	192.168.1.107
DSP 7	192.168.1.108

Table 4-1 DSP IP address table

4.4.1. Build Instruction

Steps to build web server program are listed below:

1. Import the client_evmc6678l CCS project from "Lightning_PCIE\examples\web\client\evmc6678l" directory (in CCSv5, Project->Import Existing CCS/ CCE Eclipse Projects)
2. Select DSPC8682E as active configuration.
3. Clean the client_evmc6678l project and re-build the project. After build is completed, client_evmc6678l.out and client_evmc6678l.map will be generated under "Lightning_PCIE\examples\web\client\evmc6678l\DSPC8682E" directory

4.4.2. Usage

User can use the shell script "Lightning_PCIE/examples/script/DSPC8682E/ethernet.sh" to setup Ethernet program on each DSP automatically. The following steps set up ethernet program on 8 DSPs.

1. Perform the init_1000.sh to initialize DDR
2. Perform the ethernet.sh to convert client_evmc6678l.out to client_evmc6678l.hex and load the hex file into each DSP.

```
# /Lightning_PCIE/examples/script/DSPC8682E/ethetnet.sh 8

Translating to Intel format...
   "../../web/client/evmc6678l/DSPC8682E/client_evmc6678l.out"
==> .text:_c_int00
   "../../web/client/evmc6678l/DSPC8682E/client_evmc6678l.out"  ==> .text
   "../../web/client/evmc6678l/DSPC8682E/client_evmc6678l.out"
==> .const
   "../../web/client/evmc6678l/DSPC8682E/client_evmc6678l.out"
==> .switch.1
   "../../web/client/evmc6678l/DSPC8682E/client_evmc6678l.out"  ==> .vecs
   "../../web/client/evmc6678l/DSPC8682E/client_evmc6678l.out"
==> .switch.2
   "../../web/client/evmc6678l/DSPC8682E/client_evmc6678l.out"
==> .cinit
Load HEX image: ../../../bin/DSPC8682E/client_evmc6678l.hex to 0:0, start
address 0x80300000
Load HEX OK
Load HEX image: ../../../bin/DSPC8682E/client_evmc6678l.hex to 1:0, start
address 0x80300000
Load HEX OK
Load HEX image: ../../../bin/DSPC8682E/client_evmc6678l.hex to 2:0, start
address 0x80300000
Load HEX OK
Load HEX image: ../../../bin/DSPC8682E/client_evmc6678l.hex to 3:0, start
address 0x80300000
Load HEX OK
Load HEX image: ../../../bin/DSPC8682E/client_evmc6678l.hex to 4:0, start
address 0x80300000
Load HEX OK
Load HEX image: ../../../bin/DSPC8682E/client_evmc6678l.hex to 5:0, start
address 0x80300000
Load HEX OK
Load HEX image: ../../../bin/DSPC8682E/client_evmc6678l.hex to 6:0, start
address 0x80300000
Load HEX OK
Load HEX image: ../../../bin/DSPC8682E/client_evmc6678l.hex to 7:0, start
address 0x80300000
Load HEX OK
```

3. Check the result by internet browser. The URL of DSPs are http://192.168.1.10X, X=1~8. The result is shown in Figure 4-4 and Figure 4-5.

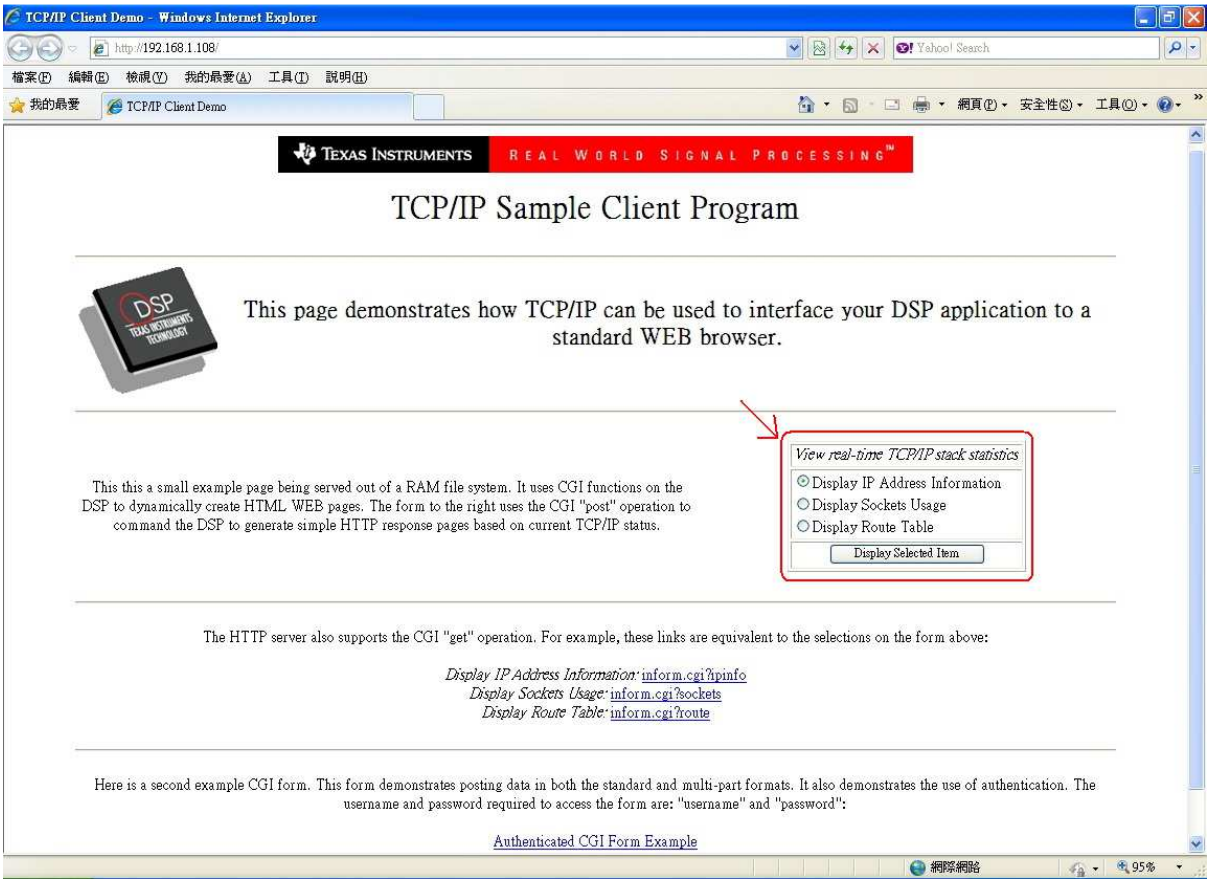
TCP/IP Client Demo - Windows Internet Explorer
http://192.168.1.108/
TCP/IP Client Demo
TEXAS INSTRUMENTS
REAL WORLD SIGNAL PROCESSING
TCP/IP Sample Client Program
This page demonstrates how TCP/IP can be used to interface your DSP application to a standard WEB browser.
This this a small example page being served out of a RAM file system. It uses CGI functions on the DSP to dynamically create HTML WEB pages. The form to the right uses the CGI "post" operation to command the DSP to generate simple HTTP response pages based on current TCP/IP status.
View real-time TCP/IP stack statistics
Display IP Address Information
Display Sockets Usage
Display Route Table
Display Selected Item
The HTTP server also supports the CGI "get" operation. For example, these links are equivalent to the selections on the form above:
Display IP Address Information: inform.cgi?ipinfo
Display Sockets Usage: inform.cgi?sockets
Display Route Table: inform.cgi?route
Here is a second example CGI form. This form demonstrates posting data in both the standard and multi-part formats. It also demonstrates the use of authentication. The username and password required to access the form are: "username" and "password":
Authenticated CGI Form Example

Figure 4-4 TCP/IP Demo Page

http://192.168.1.108/inform.cgi - Windows Internet Explorer
TEXAS INSTRUMENTS
REAL WORLD SIGNAL PROCESSING
IP Address Information
HTTP Server IP Address 192.168.1.108
HTTP Server Hostname No DNS Reply
Your IP Address 192.168.1.1
Your Hostname No DNS Reply
Return to Main Page

Figure 4-5 IP Address Information page

4.5. Communication between PC and DSP

This example demonstrates several functions for manipulating the DSPs including:

1. Write data blocks to DSP memory from PC
2. Read back data blocks from DSP memory to PC
3. PC interrupts DSP
4. DSP interrupts PC
5. Emulate “console” output. The implementation enables the DSP to display messages to PC. This could be helpful when developing and debugging DSP applications.

4.5.1. Build Instruction

Steps to build ipc DSP program are listed below:

1. Import the demo_evmc6678l CCS project from "Lightning_PCIE\examples\ipc\dsp\evmc6678l" directory (in CCSv5, Project->Import Existing CCS/ CCE Eclipse Projects)

2. Select DSPC8682E as active configuration.

3. Clean the demo_evmc6678l project and re-build the project. After build is completed, demo_evm6678l.out and demo_evm6678l.map will be generated under "Lightning_PCIE\examples\ipc\dsp\evmc6678l\ bin\DSPC8682E" directory

4.5.2. Usage

User can use shell script file "Lightning_PCIE/examples/script/DSPC8682E/ipc.sh" to load demo_evm6678l.hex into the specific DSP. The script will do following four jobs.

1. Convert .out to .hex.
2. Load demo_evm6678l.hex to the specified DSP.
3. Run the PC – DSP intercommunication demo (repeat 1000 times).
4. Run the console output demo.

There two Steps to launch IPC example,

1. Perform init_1000.sh to initialize DDR
2. Run ipc.sh

The following example captures the result of running " Lightning_PCIE/examples/script/DSPC8682E/ipc.sh ". Refer to 4.5.4 (DSP Demo Program) to get detailed procedure of the DSP demo program.

```
# /Lightning_PCIE/examples/script/DSPC8682E/ipc.sh 1

Translating to Intel format...
   "../../ipc/dsp/evmc6678l/bin/DSPC8682E/demo_evm6678l.out"
==> .text:_c_int00
   "../../ipc/dsp/evmc6678l/bin/DSPC8682E/demo_evm6678l.out"   ==> .text
   "../../ipc/dsp/evmc6678l/bin/DSPC8682E/demo_evm6678l.out"   ==> .const
   "../../ipc/dsp/evmc6678l/bin/DSPC8682E/demo_evm6678l.out"   ==> .csl_vect
   "../../ipc/dsp/evmc6678l/bin/DSPC8682E/demo_evm6678l.out"   ==> .switch
   "../../ipc/dsp/evmc6678l/bin/DSPC8682E/demo_evm6678l.out"   ==> .cinit
Load HEX image: ../../../bin/DSPC8682E/demo_evm6678l.hex to 1:0, start address
0x00840000
Load HEX OK
DDR of DSP is initialized, ready to write dummy data to DSP
dump dummy_buffer before DSP operation:

0x607180      00000000 00000001 00000002 00000003 00000004 00000005 00000006 00000007
0x6071a0      00000008 00000009 0000000a 0000000b 0000000c 0000000d 0000000e 0000000f
0x6071c0      00000010 00000011 00000012 00000013 00000014 00000015 00000016 00000017
0x6071e0      00000018 00000019 0000001a 0000001b 0000001c 0000001d 0000001e 0000001f
0x607200      00000020 00000021 00000022 00000023 00000024 00000025 00000026 00000027
```

```
0x607220      00000028 00000029 0000002a 0000002b 0000002c 0000002d 0000002e 0000002f
0x607240      00000030 00000031 00000032 00000033 00000034 00000035 00000036 00000037
0x607260      00000038 00000039 0000003a 0000003b 0000003c 0000003d 0000003e 0000003f
0x607280      00000040 00000041 00000042 00000043 00000044 00000045 00000046 00000047
0x6072a0      00000048 00000049 0000004a 0000004b 0000004c 0000004d 0000004e 0000004f
0x6072c0      00000050 00000051 00000052 00000053 00000054 00000055 00000056 00000057
0x6072e0      00000058 00000059 0000005a 0000005b 0000005c 0000005d 0000005e 0000005f
0x607300      00000060 00000061 00000062 00000063 00000064 00000065 00000066 00000067
0x607320      00000068 00000069 0000006a 0000006b 0000006c 0000006d 0000006e 0000006f
0x607340      00000070 00000071 00000072 00000073 00000074 00000075 00000076 00000077
0x607360      00000078 00000079 0000007a 0000007b 0000007c 0000007d 0000007e 0000007f
0x607380      00000080 00000081 00000082 00000083 00000084 00000085 00000086 00000087
0x6073a0      00000088 00000089 0000008a 0000008b 0000008c 0000008d 0000008e 0000008f
0x6073c0      00000090 00000091 00000092 00000093 00000094 00000095 00000096 00000097
0x6073e0      00000098 00000099 0000009a 0000009b 0000009c 0000009d 0000009e 0000009f
receive interrupt from dsp1
dummy data has already been changed by DSP
dump dummy_buffer after DSP operation:

0x607180      00000001 00000002 00000003 00000004 00000005 00000006 00000007 00000008
0x6071a0      00000009 0000000a 0000000b 0000000c 0000000d 0000000e 0000000f 00000010
0x6071c0      00000011 00000012 00000013 00000014 00000015 00000016 00000017 00000018
0x6071e0      00000019 0000001a 0000001b 0000001c 0000001d 0000001e 0000001f 00000020
0x607200      00000021 00000022 00000023 00000024 00000025 00000026 00000027 00000028
0x607220      00000029 0000002a 0000002b 0000002c 0000002d 0000002e 0000002f 00000030
0x607240      00000031 00000032 00000033 00000034 00000035 00000036 00000037 00000038
0x607260      00000039 0000003a 0000003b 0000003c 0000003d 0000003e 0000003f 00000040
0x607280      00000041 00000042 00000043 00000044 00000045 00000046 00000047 00000048
0x6072a0      00000049 0000004a 0000004b 0000004c 0000004d 0000004e 0000004f 00000050
0x6072c0      00000051 00000052 00000053 00000054 00000055 00000056 00000057 00000058
0x6072e0      00000059 0000005a 0000005b 0000005c 0000005d 0000005e 0000005f 00000060
0x607300      00000061 00000062 00000063 00000064 00000065 00000066 00000067 00000068
0x607320      00000069 0000006a 0000006b 0000006c 0000006d 0000006e 0000006f 00000070
0x607340      00000071 00000072 00000073 00000074 00000075 00000076 00000077 00000078
0x607360      00000079 0000007a 0000007b 0000007c 0000007d 0000007e 0000007f 00000080
0x607380      00000081 00000082 00000083 00000084 00000085 00000086 00000087 00000088
0x6073a0      00000089 0000008a 0000008b 0000008c 0000008d 0000008e 0000008f 00000090
0x6073c0      00000091 00000092 00000093 00000094 00000095 00000096 00000097 00000098
0x6073e0      00000099 0000009a 0000009b 0000009c 0000009d 0000009e 0000009f 000000a0
.
.
.
Synchronizing ... done.
============================
PCIe Hello World Example, this is DSP1
Debug: GEM-INTC Configuration Completed
Debug: CPINTC-0 Configuration...
Debug: CPINTC-0 Configuration Completed
```

4.5.3. PC Site Utility

The PC site demo application dsp_demo contains two commands, demo and console function.

4.5.3.1. Inter Communication

The demo command is used to demonstrate the negotiation between DSP and PC host. dsp_demo will perform the data blocks read/write and wait the interrupt signal which is sent from PCIe driver.

The command syntax is:

dsp_demo demo [chip#]

[chip#] (the number of DSPs) parameter selects which DSP will be accessed by PC.

4.5.3.2. Console Simulation

This command is for creating a virtual console to display the debug message by the program running in specific DSP (chip# from DSP#0 to DSP#7) and cores (core# from CPU core#0 to CPU core#7).

The command syntax is:

dsp_demo console [chip#] [core#]

The following example displays the debug message of demo_evm6678l.hex (DSP demo program) which is executed by CPU#0 in DSP#0.

```
# /Lightning_PCIE/examples/ipc/pc/bin/dsp_demo console 0 0

Synchronizing ... done.
============================
PCIe Hello World Example, this is DSP0
Debug: GEM-INTC Configuration Completed
Debug: CPINTC-0 Configuration...
Debug: CPINTC-0 Configuration Completed
DSP0 generated interrupt to host
DSP0 receive MSI interrupt from host
DSP0 finish operating dummy data.
```

Note: DSP program demo_evm6678l.hex should be downloaded to DSP device first before issuing this virtual console command. Refer to the source code of DSP demo program to get detailed implementation.

4.5.4. DSP Demo Program

DSP demo program configures DSP CSL INTC registers to receive PCIe Legacy INTB and MSI0 interrupt from PC host. The procedure of demo example is illustrated below with flow chart displayed in Figure 4-6:

1. DSP application set up INTC for ISR handler to receive Legacy INTB and MSI0, then wait the interrupt sent from PC host

2. PC host writes test data pattern whose length is 640-byte to DSP DDR and sends an interrupt to DSP after finishing the writing of the test data pattern
3. The test data pattern in DDR will be added by 1 when DSP receives the interrupt from PC host. After finishing the operation, DSP will send an interrupt back to PC host.
4. PC Host receives the interrupt from DSP as the indication that the test data pattern has already been changed and prints the test data pattern
5. Repeat the communication 1000 times.

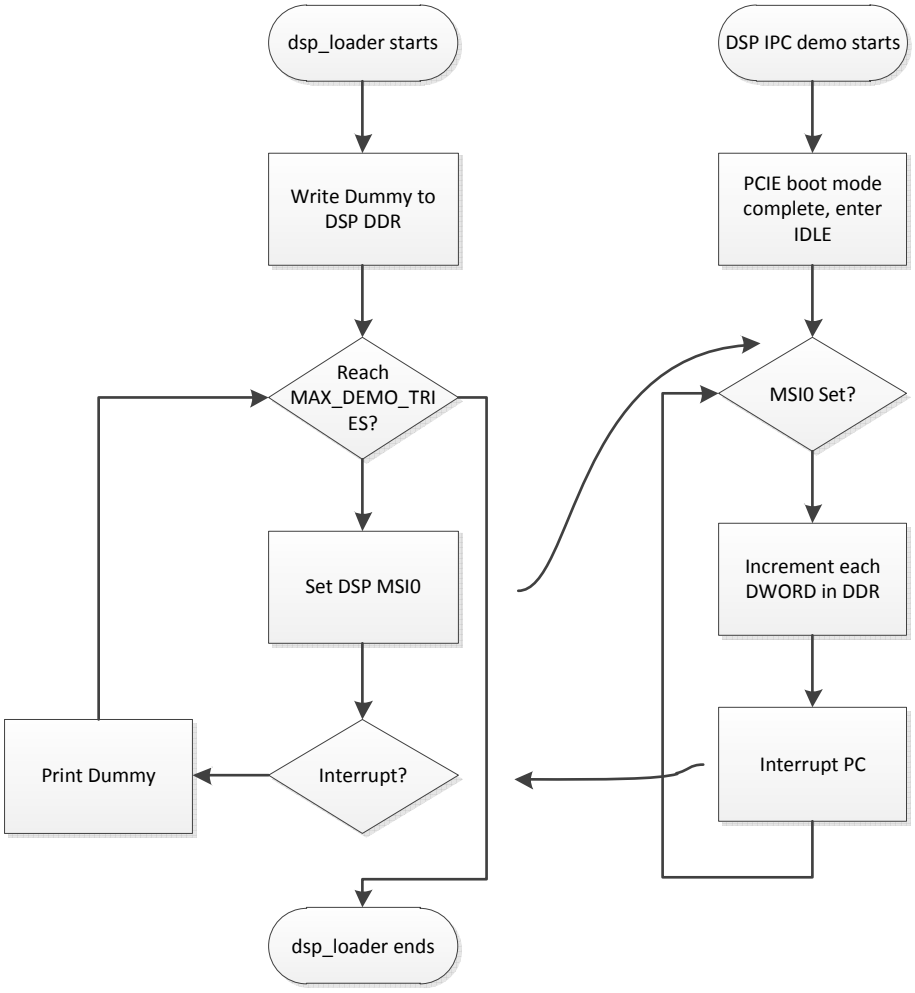

Figure 4-6 Flow Diagram of IPC Example

Besides the interrupt demo, the demo code also contains the virtual console implementation and the debug message will be written into L2 memory. PC host can use dsp_demo console command to dump these messages for debug purpose.

4.5.5. IPC Demo on SYS/BIOS DSP Application

The IPC feature also works in TI's SYS/BIOS architecture. The demo program is embedded in the Ethernet example to show how to register an interrupt in SYS/BIOS architecture. In order to run this IPC demo, users can follow two steps below.

1. Initialize DDR3 module (perform init_1000.sh).
2. Run SYS/BIOS IPC demo script (perform ipc_SYSBIOS.sh)

```
# /Lightning_PCIE/examples/script/DSPC8682E/ipc_SYSBIOS.sh 1

Translating to Intel format...
   "../../web/client/evmc6678l/DSPC8682E/client_evmc6678l.out"
==> .text:_c_int00
   "../../web/client/evmc6678l/DSPC8682E/client_evmc6678l.out"   ==> .text
   "../../web/client/evmc6678l/DSPC8682E/client_evmc6678l.out"   ==> .const
   "../../web/client/evmc6678l/DSPC8682E/client_evmc6678l.out"
==> .switch.1
   "../../web/client/evmc6678l/DSPC8682E/client_evmc6678l.out"   ==> .vecs
   "../../web/client/evmc6678l/DSPC8682E/client_evmc6678l.out"
==> .switch.2
   "../../web/client/evmc6678l/DSPC8682E/client_evmc6678l.out"   ==> .cinit
Load HEX image: ../../../bin/DSPC8682E/client_evmc6678l.hex to 1:0, start
address 0x80300000
Load HEX OK
DDR of DSP is initialized, ready to write dummy data to DSP
dump dummy_buffer before DSP operation:

0x607180      00000000 00000001 00000002 00000003 00000004 00000005 00000006 00000007
0x6071a0      00000008 00000009 0000000a 0000000b 0000000c 0000000d 0000000e 0000000f
0x6071c0      00000010 00000011 00000012 00000013 00000014 00000015 00000016 00000017
0x6071e0      00000018 00000019 0000001a 0000001b 0000001c 0000001d 0000001e 0000001f
0x607200      00000020 00000021 00000022 00000023 00000024 00000025 00000026 00000027
0x607220      00000028 00000029 0000002a 0000002b 0000002c 0000002d 0000002e 0000002f
0x607240      00000030 00000031 00000032 00000033 00000034 00000035 00000036 00000037
0x607260      00000038 00000039 0000003a 0000003b 0000003c 0000003d 0000003e 0000003f
0x607280      00000040 00000041 00000042 00000043 00000044 00000045 00000046 00000047
0x6072a0      00000048 00000049 0000004a 0000004b 0000004c 0000004d 0000004e 0000004f
0x6072c0      00000050 00000051 00000052 00000053 00000054 00000055 00000056 00000057
0x6072e0      00000058 00000059 0000005a 0000005b 0000005c 0000005d 0000005e 0000005f
0x607300      00000060 00000061 00000062 00000063 00000064 00000065 00000066 00000067
0x607320      00000068 00000069 0000006a 0000006b 0000006c 0000006d 0000006e 0000006f
0x607340      00000070 00000071 00000072 00000073 00000074 00000075 00000076 00000077
0x607360      00000078 00000079 0000007a 0000007b 0000007c 0000007d 0000007e 0000007f
0x607380      00000080 00000081 00000082 00000083 00000084 00000085 00000086 00000087
0x6073a0      00000088 00000089 0000008a 0000008b 0000008c 0000008d 0000008e 0000008f
0x6073c0      00000090 00000091 00000092 00000093 00000094 00000095 00000096 00000097
0x6073e0      00000098 00000099 0000009a 0000009b 0000009c 0000009d 0000009e 0000009f
receive interrupt from dsp1
dummy data has already been changed by DSP
dump dummy_buffer after DSP operation:

0x607180      00000001 00000002 00000003 00000004 00000005 00000006 00000007 00000008
0x6071a0      00000009 0000000a 0000000b 0000000c 0000000d 0000000e 0000000f 00000010
```

```
0x6071c0    00000011 00000012 00000013 00000014 00000015 00000016 00000017 00000018
0x6071e0    00000019 0000001a 0000001b 0000001c 0000001d 0000001e 0000001f 00000020
0x607200    00000021 00000022 00000023 00000024 00000025 00000026 00000027 00000028
0x607220    00000029 0000002a 0000002b 0000002c 0000002d 0000002e 0000002f 00000030
0x607240    00000031 00000032 00000033 00000034 00000035 00000036 00000037 00000038
0x607260    00000039 0000003a 0000003b 0000003c 0000003d 0000003e 0000003f 00000040
0x607280    00000041 00000042 00000043 00000044 00000045 00000046 00000047 00000048
0x6072a0    00000049 0000004a 0000004b 0000004c 0000004d 0000004e 0000004f 00000050
0x6072c0    00000051 00000052 00000053 00000054 00000055 00000056 00000057 00000058
0x6072e0    00000059 0000005a 0000005b 0000005c 0000005d 0000005e 0000005f 00000060
0x607300    00000061 00000062 00000063 00000064 00000065 00000066 00000067 00000068
0x607320    00000069 0000006a 0000006b 0000006c 0000006d 0000006e 0000006f 00000070
0x607340    00000071 00000072 00000073 00000074 00000075 00000076 00000077 00000078
0x607360    00000079 0000007a 0000007b 0000007c 0000007d 0000007e 0000007f 00000080
0x607380    00000081 00000082 00000083 00000084 00000085 00000086 00000087 00000088
0x6073a0    00000089 0000008a 0000008b 0000008c 0000008d 0000008e 0000008f 00000090
0x6073c0    00000091 00000092 00000093 00000094 00000095 00000096 00000097 00000098
0x6073e0    00000099 0000009a 0000009b 0000009c 0000009d 0000009e 0000009f 000000a0
```

4.6. Image Processing Demonstration

The image processing program is modified from the example codes in TI MCSDK. This application shows implementation of an image processing system using a simple multicore framework. This application will run TI image processing kernels (imagelib) on multiple cores to do image processing (eg: edge detection, etc) on an input image.

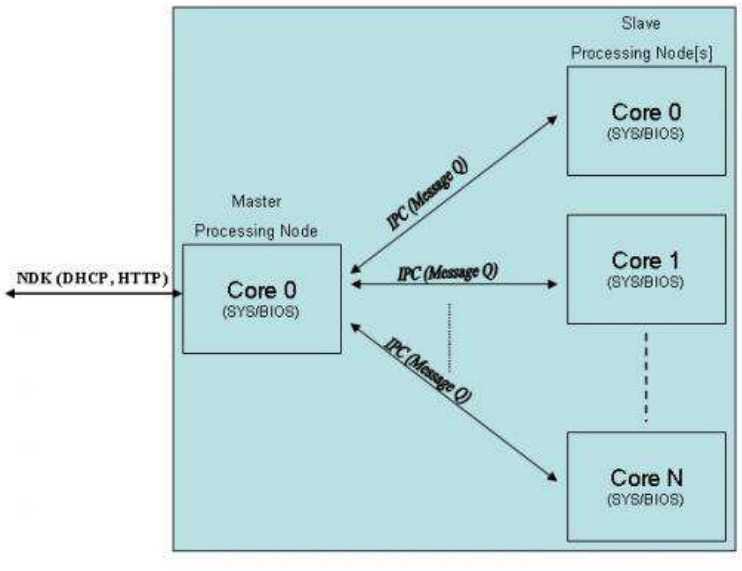

Figure 4-7 Image Processing Application Software Framework

The user input image will be BMP image. The image will be transferred to external memory using NDK (http). The Ethernet port on the bracket of the DSPC8682E must be connected to an external Ethernet switch (support gigabit rates) before running this example. Each DSP

has a fixed IP number that is determined by its order. The pre-given IP addresses are shown below. The user can use a browser to input the BMP image form web page provided by HTTP server.

	IP
DSP 0	192.168.1.101
DSP 1	192.168.1.102
DSP 2	192.168.1.103
DSP 3	192.168.1.104
DSP 4	192.168.1.105
DSP 5	192.168.1.106
DSP 6	192.168.1.107
DSP 7	192.168.1.108

PC Setting	
IP	192.168.1.100
Subnet Mask	255.255.254.0

4.6.1. Build Instruction

Steps to build image processing program are listed below:

1. Import the image_processing_evmc6678l_master and image_processing_evmc6678l_slave CCS projects from "Lightning_PCIE\examples\image_processing\ipc\evmc6678l" directory (in CCSv5, Project->Import Existing CCS/ CCE Eclipse Projects)
2. Select DSPC8682E as active configuration.
3. Clean the image_processing_evmc6678l_master project and re-build the project. After build is completed, image_processing_evmc6678l_master.out will be generated under the directory:

   "Lightning_PCIE\examples\image_processing\ipc\evmc6678l\master\DSPC8682E\"
4. Clean the image_processing_evmc6678l_slave project and re-build the project. After build is completed, image_processing_evmc6678l_slave.out will be generated under the directory: "Lightning_PCIE\examples\image_processing\ipc\evmc6678l\slave\Debug"

4.6.2. Usage

User can use the shell script "Lightning_PCIE/examples/script/DSPC8682E/image_processing.sh" to setup image

processing program on each DSP automatically. The following steps set up image processing program on 8 DSPs.

1. Perform init_1000.sh to initialize DDR
2. Run image_processing.sh, it will convert .out file to .hex and load the images to each DSP.

```
# /Lightning_PCIE/examples/script/DSPC8682E/image_processing.sh 8

Translating to Intel format...

"../../image_processing/ipc/evmc6678l/slave/Debug/image_processing_evmc66
78l_slave.out"   ==> .text:_c_int00

"../../image_processing/ipc/evmc6678l/slave/Debug/image_processing_evmc66
78l_slave.out"   ==> .text

"../../image_processing/ipc/evmc6678l/slave/Debug/image_processing_evmc66
78l_slave.out"   ==> .const

"../../image_processing/ipc/evmc6678l/slave/Debug/image_processing_evmc66
78l_slave.out"   ==> .switch

"../../image_processing/ipc/evmc6678l/slave/Debug/image_processing_evmc66
78l_slave.out"   ==> .vecs

"../../image_processing/ipc/evmc6678l/slave/Debug/image_processing_evmc66
78l_slave.out"   ==> .cinit
Translating to Intel format...

"../../image_processing/ipc/evmc6678l/master/DSPC8682E/image_processing_e
vmc6678l_master.out"   ==> .text:_c_int00

"../../image_processing/ipc/evmc6678l/master/DSPC8682E/image_processing_e
vmc6678l_master.out"   ==> .text

"../../image_processing/ipc/evmc6678l/master/DSPC8682E/image_processing_e
vmc6678l_master.out"   ==> .const.1

"../../image_processing/ipc/evmc6678l/master/DSPC8682E/image_processing_e
vmc6678l_master.out"   ==> .const.2

"../../image_processing/ipc/evmc6678l/master/DSPC8682E/image_processing_e
vmc6678l_master.out"   ==> .switch.1

"../../image_processing/ipc/evmc6678l/master/DSPC8682E/image_processing_e
vmc6678l_master.out"   ==> .vecs

"../../image_processing/ipc/evmc6678l/master/DSPC8682E/image_processing_e
vmc6678l_master.out"   ==> .switch.2
```

```
"../../image_processing/ipc/evmc6678l/master/DSPC8682E/image_processing_e
vmc6678l_master.out"   ==> .cinit
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 0:1,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 0:2,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 0:3,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 0:4,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 0:5,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 0:6,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 0:7,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 1:1,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 1:2,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 1:3,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 1:4,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 1:5,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 1:6,
start address 0x0c100000
```

```
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 1:7,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 2:1,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 2:2,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 2:3,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 2:4,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 2:5,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 2:6,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 2:7,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 3:1,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 3:2,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 3:3,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 3:4,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 3:5,
start address 0x0c100000
Load HEX OK
```

```
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 3:6,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 3:7,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 4:1,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 4:2,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 4:3,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 4:4,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 4:5,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 4:6,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 4:7,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 5:1,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 5:2,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 5:3,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 5:4,
start address 0x0c100000
Load HEX OK
Load HEX
```

```
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 5:5,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 5:6,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 5:7,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 6:1,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 6:2,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 6:3,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 6:4,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 6:5,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 6:6,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 6:7,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 7:1,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 7:2,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 7:3,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 7:4,
```

```
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 7:5,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 7:6,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_slave.hex to 7:7,
start address 0x0c100000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_master.hex to 0:0,
start address 0x0c000000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_master.hex to 1:0,
start address 0x0c000000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_master.hex to 2:0,
start address 0x0c000000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_master.hex to 3:0,
start address 0x0c000000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_master.hex to 4:0,
start address 0x0c000000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_master.hex to 5:0,
start address 0x0c000000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_master.hex to 6:0,
start address 0x0c000000
Load HEX OK
Load HEX
image: ../../../bin/DSPC8682E/image_processing_evmc6678l_master.hex to 7:0,
start address 0x0c000000
Load HEX OK
```

3. Please refer to the Figure 4-8. Input the BMP image form the internet browser. The URL of DSPs are http://192.168.1.10X, X=1~8. Select the number of core and image path for processing.

4. The output result is shown in Figure 4-9.

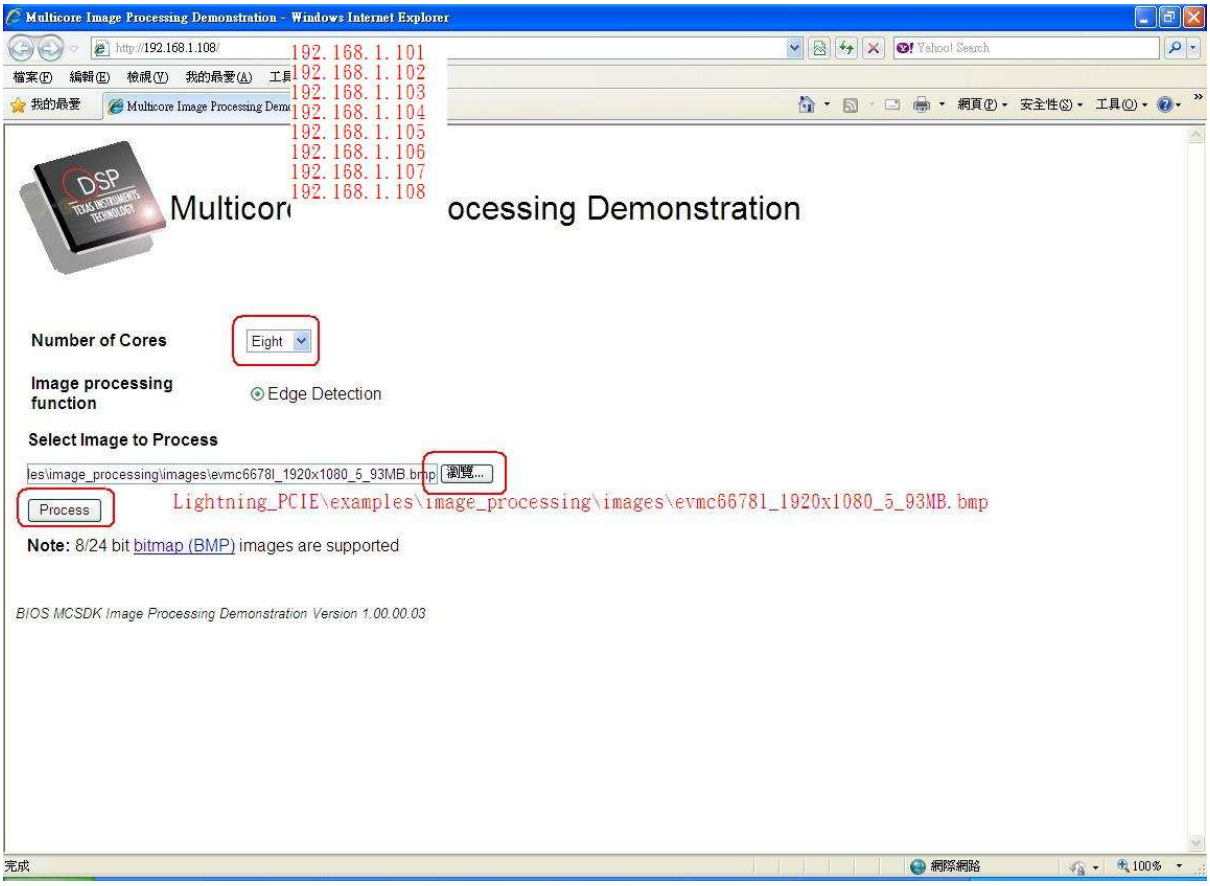

Figure 4-8 Image Processing Input Page

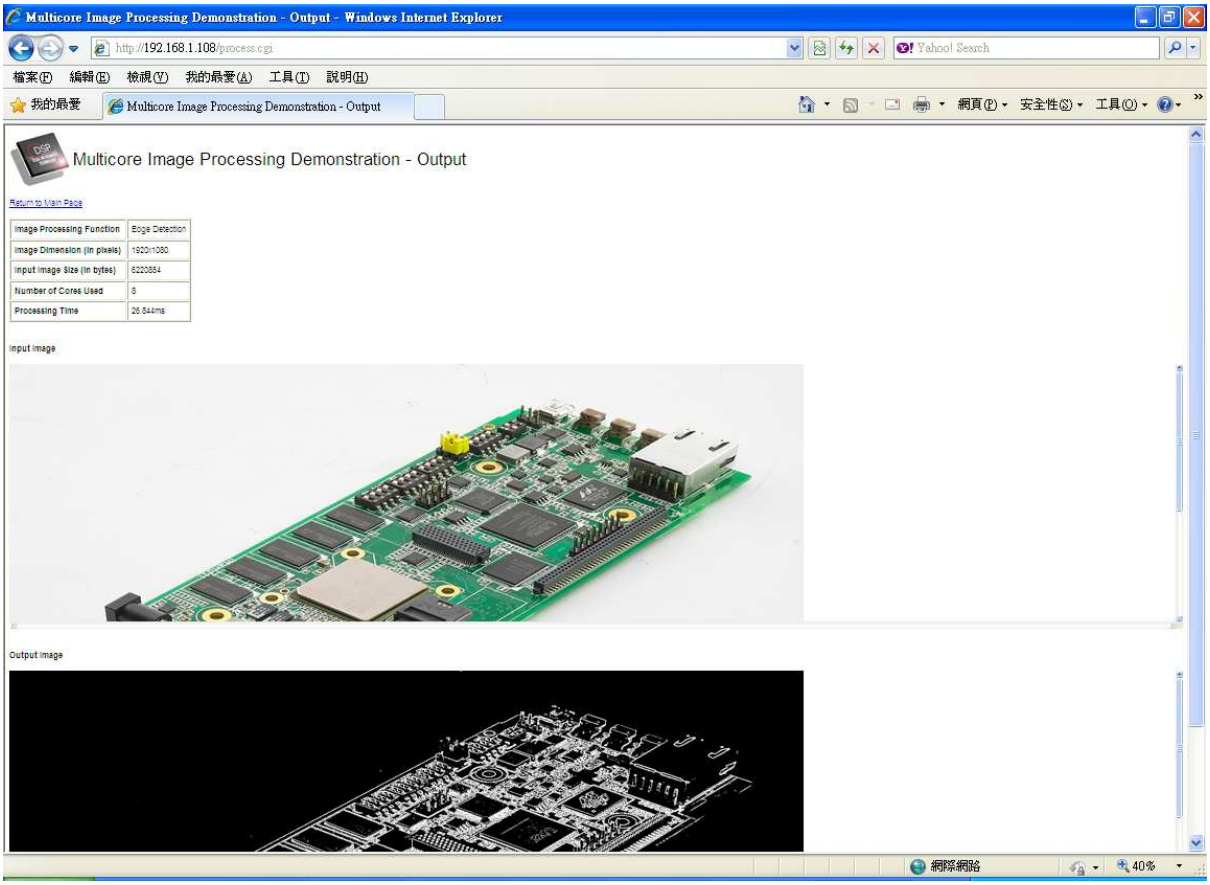

Figure 4-9 Image Processing Output Page